PATENT

C. REMARKS

1. Status of the Claims

Claims 1-7, 9-17, 19-28, and 30-32 are currently present in the Application and stand rejected. Claims 1, 12, and 22 are independent claims and have been amended along with claims 2, 3, 5, 9, 13-15, 19, 22-24, 26, and 30. Claims 8, 18, and 29 have been cancelled with the limitations originally found in these dependent claims being incorporated in their respective independent claims. No claims have been added in this Response.

2. Summary of the Present Invention

Applicants' invention is a method / system / program product that manages a shared resource by determining whether a process that is requesting the resource is requesting read access or write access to the resource. The process allows a single read requestor to write to the resource or allows multiple readers to read from the resource simultaneously. In addition, as amended, the independent claims each claim "speeding up" of one or more of the read requestors that acquire the lock. Applicants' attorney explained in the Examiner Interview of December 22, 2003, that when multiple read requestors acquire a lock in "read only" mode, a process waiting to write to the resource needs to wait until *all* of the readers are finished. This can be problematic if one or more of the readers has a low system priority and, therefore, is time sliced or otherwise takes longer in order to finish its read processing. One way to speed up these slower processes is by providing them with a temporary time slice exemption (claim 9) so that the low priority process will not be time sliced and

will therefore be able to finish its read processing more expeditiously.

Two other claims discussed during the Examiner Interview on December 22, 2003, were claims 10 and 11. Claim 10 is a claim directed to identifying a processes that are "upgraders." Upgraders are defined as processes that are requesting to atomically upgrade a shared resource. This typically happens when a reader of a shared resource wants to write a new value to the resource based upon the value that was read so that the value being written is consistent with the value currently being read. In this instance, the "reader" becomes a "writer" without allowing any interleaving writers to alter the current value of the resource (see Figure 7 and pages 28-30 of the specification for further details). Claim 10 deals with identifying an upgrader in the queue and granting the upgrader write access to the shared resource, while Claim 11 (depending upon Claim 10), deals with speeding up the upgrader process by boosting the process' priority.

3. Formal Drawings

In Applicants' response of February 3, 2004, Applicants requested that the Examiner indicate whether Applicants' formal drawings have been accepted. Applicants note that the Examiner did not indicate that Applicants' drawings are accepted and, once again, respectfully request such an indication in the next communication.

4. Claim Rejections - 35 U.S.C. § 103 - Alleged Obviousness

Claims 1-4, 12-14, and 22-25 were rejected under 35 U.S.C. § 103 as allegedly being obvious over U.S. Patent No. 5,761,659 to Bertoni et al. (hereinafter "Bertoni"). Applicants have amended each of the independent claims 1, 12, and 22 to include limitations originally found in claims 8, 18, and 29, respectively. These claims were rejected as allegedly being obvious over Bertoni in view of U.S. Patent No. 5,490,270 to Devarakonda et al. (hereinafter "Devarakonda"). Applicants respectfully traverse the rejections.

As amended, each of Applicants' independent claims include the limitations of:

- determining whether a process identifier included in a queue corresponds to a read requestor or a write requestor;
- allowing the write requestor to write to the shared resource in response to the process identifier corresponding to the write requestor; ~~and~~
- allowing one or more successive read requestors to read from the shared resource in response to the process identifier corresponding to one of the read requestors; and
    - speeding up processing for one or more of the read requestors that acquire the resource lock.

The Office Action alleges that Bertoni teaches a method of managing a shared resource by determining whether a process identifier included in a queue corresponds to a read requestor or a write requestor, allowing a write requestor to write to the shared resource in response to the process identifier corresponding to the write requestor, and allowing one or more successive read requestors to read from the shared resource in response to the process identifier corresponding to one of the

read requestors. The Office Action admits that Bertoni does not teach or suggest using a process identifier to locate a process in the queue, but the Office Action asserts that one of ordinary skill could have modified Bertoni's system to include a process identifier so that the different processes could be uniquely identified. While Applicants do not agree with the assumption made in the Office Action, Applicants would rather focus on the fact that the art of record does not teach or suggest "speeding up processing for one or more read requestors that acquire the resource lock."

The Office Action admits that Bertoni does not teach or suggest Applicants' claimed "speeding up" of read requestors. However, the Office Action alleges that Devarakonda teaches such limitation. Applicants respectfully disagree with this allegation because nowhere does Devarakonda teach or suggest speeding up of requestors that acquire a lock to a resource. The Office Action alleges that Devarakonda teaches this limitation at column 4, lines 11-30. This section of Devarakonda has been reproduced below:

> When an LLM wants to upgrade a token (e.g. it has a READ token and it wants a WRITE token or it has a WRITE-SHARED token and it wants a WRITE-EXCLUSIVE token) it sends an UPGRADE request to the LCS and awaits a reply. Upon receipt of the reply, the LLM checks to see if the token has been upgraded. If the token has been upgraded, the LLM updates the state of the token in its table and thus complete the token upgrade. If the token has not been upgraded, the reply from the LCS contains the copyset of current token holders and in response, the LLM sends a revoke messages to the copyset nodes.
>
> After receiving acknowledgement from the copyset nodes, the LLM informs the LCS that the token has been revoked. In response, the LCS changes the LCS state table entry for the token to reflect that the requesting LLM is the only holder of the token and the token is in the requested mode. The LCS then sends an acknowledgement to the requesting LLM.

> After receiving an acknowledgement, the LLM updates the state of the token in its own state table 904 and thus completes the token upgrade.

In the cited section, Devarakonda teaches that an LLM (local lock manager) can "upgrade" a token (i.e., a lock). Devarakonda describes a typical situation being where a lock is held in "READ" mode and the LLM desires a "WRITE" mode access to the lock or if the LLM has "WRITE-SHARED" access to the lock and desires "WRITE-EXCLUSIVE" access. Devarakonda goes on to explain how the LLM responds if the upgrade request is successful as well as what steps are performed if the upgrade did not take place (i.e., Devarakonda explains that "revoke" messages are sent to various nodes so that the lock can be acquired in an upgraded fashion.). In essence, Devarakonda is simply teaching that the holder of a resource lock can upgrade the lock so that instead of simply reading from the lock, the process can upgrade to become a "writer" and write information to the lock. Upgrading requests are common in the prior art, but importantly, have nothing to do whatsoever in speeding up the requestor process, as taught and claimed by Applicants. A common upgrade situation is where a process wants to read a value in a shared resource and, depending on the value of the shared resource, may leave the value alone or may change the value. In this situation, the process first acquires a "read" lock to the resource so that it can read the value. After reading the value, if the process needs to update the value, the process will request to UPGRADE its lock so that it can write to the shared resource (i.e., the READER becomes a WRITER). Moving from a "reader" to a "writer" consumes resources with the process that makes the UPGRADE request often having to wait until other processes have released their holds on the lock and

the UPGRADE request can be satisfied. In other words, becoming an UPGRADER as described by Devarakonda typically ***slows down*** a process as the process needs to wait for its upgrade request to be satisfied so that it gets the upgraded access to the lock and can perform its actions (i.e., write) to the shared resource.

On the other hand, Applicants teach and claim a way of "speeding up processing for one or more of the read requestors that acquire the resource lock" in each of the independent claims. One way such "speeding up" can be accomplished is by granting the readers a "temporary time slice exemption," as claimed by Applicants in claims 9, 19, and 20. As explained above, speeding up readers that acquire a lock typically enables the readers to finish using the lock faster and, thus, release the lock sooner so that it can be used by another process or other processes.

The Office Action alleges that Devarakonda teaches speeding up reader processes by granting time slice exemptions to readers, citing the same section (col. 4, lines 11-30) that Applicants have reproduced above. However, upon inspection of the sited portion of Devarakonda, it is clear not only that Devarakonda is not dealing with speeding up readers (as discussed above), but also that Devarakonda never mentions granting any process, let alone the reader that is requesting an UPGRADE in Devarakonda, a time slice exemption. The Office Action admits that Bertoni does not teach or suggest speeding up readers that acquire a lock and, upon review of Bertoni, it is clear that no such teaching exists in Bertoni. As explained above, Devarakonda is completely void of such teaching as Devarakonda (1) is teaching how a process that has a read lock becomes an UPGRADER in order to write to the lock, and (2) that

Devarakonda does not teach speeding up processes that acquire the lock (in fact, as explained above, becoming an UPGRADER typically slows down a process as it has to first become a reader of the shared resource and then become a writer of the shared resource). As shown above, neither Bertoni nor Devarakonda, taken alone or in combination with one another, teach or suggest Applicants' claimed invention that includes determining whether a process identifier included in a queue corresponds to a read requestor or a write requestor; allowing the write requestor to write to the shared resource in response to the process identifier corresponding to the write requestor; and allowing one or more successive read requestors to read from the shared resource in response to the process identifier corresponding to one of the read requestors; and ***speeding up processing*** for one or more of the read requestors that acquire the resource lock, as claimed by Applicants in each of the independent claims 1, 12, and 22. Therefore, each of the independent claims is allowable over Bertoni in view of Devarakonda.

Claims 2-7 and 9-11 each depend on claim 1 and, therefore, are allowable for at least the same reason that claim 1 is allowable. Claims 13-17 and 19-21 each depend on claim 12 and, therefore, are allowable for at least the same reason that claim 12 is allowable. Claims 23-28 and 30-32 each depend on claim 22 and, therefore, are allowable for at least the same reason that claim 22 is allowable.

Claims 10, 11, 20, 21, 31, and 32 were rejected under 35 U.S.C. § 103 as allegedly being unpatentable over Bertoni in view of U.S. Patent No. 5,872,909 to Wilner et al. (hereinafter "Wilner"). While these claims are allowable because they depend

upon allowable independent claims 1, 12, and 22, as explained above, these claims are also allowable because Wilner does not teach or suggest "identifying" and "granting" an upgrader a write lock to a shared resource (claims 10, 20, and 31), nor does Wilner teach or suggest "boosting a priority" of the upgrader prior to the upgrader writing to the shared resource. The section of Wilner cited in the Office Action in support of this allegation (col. 5, lines 21-57) is reproduced below:

> Task states include "delayed," "executing," "inherited," "locked," "pended," "ready" or "suspended." A task is delayed if it has been explicitly directed to pause for an amount of time. A task or ISR is executing if the task or ISR has control of the processor. In a preferred embodiment, for a task to be in the executing state, there must be no interrupts to service, the task must be the highest-priority, ready task in the system, and there must be no other tasks with preemption locking enabled. ISRs are in the executing state after their interrupt has been acknowledged by the kernel. If there is more than one ISR to service, the one at the processor's higher interrupt level executes. The idle loop is in the executing state when there no tasks to run and no ISRs to service.
>
> Each task has a "priority" which indicates that task's eligibility to control the CPU relative to the other tasks in the system. A task is in the inherited state when its priority has been increased because it owns a mutual exclusion semaphore that has priority inheritance enabled and a higher-priority task is waiting for that semaphore. Priority inheritance is a solution to the priority inversion problem: a higher-priority task being forced to wait an indefinite period of time for the completion of a lower-priority task. For example, assume that task 30 needs to take the mutual exclusion semaphore for a region, but taskLow currently owns the semaphore. Although taskHi preempts taskLow, taskHi pends immediately because it cannot take the semaphore. Under some circumstances, taskLow can then run, complete its critical region and release the semaphore, making taskHi ready to run. If, however, a third task, taskMed, preempts taskLow such that taskLow is unable to release the semaphore, then taskHi will never get to run. With the priority inheritance option enabled, a task, such as taskLow, that owns a resource executes at the priority of the highest-priority task pended on that resource, the priority of taskHi in the

example. When the task gives up the resource, it returns to its normal priority.

In the section cited in the Office Action, Wilner teaches that a common problem faced when managing multiple processes is the "priority inversion problem" where a high priority task must wait for a lower priority task to complete its work. However, Wilner is discussing the problem of processes acquiring resources and is **not** directed at a particular type of process claimed by Applicants (the UPGRADER). The UPGRADER, as claimed by Applicants, is a process that reads a resource and, based on the value of the resource, may need to write to the resource immediately (i.e., so that no other process can alter the value of the resource until after the UPGRADER has altered the value). Because the process that requests an UPGRADE needs to write to the resource, a "WRITE" lock is granted to the resource (claims 10, 20, and 31). In order for the UPGRADER to complete the writing quickly, Applicants claim boosting the upgrader's priority (claims 11, 21, and 32). On the other hand, Wilner simply teaches how low- and high-priority processes vie for shared resources. Wilner simply does not teach or suggest how a process becomes an UPGRADER of a shared resource. It follows, therefore, that Wilner also does not teach or suggest boosting the priority of an UPGRADER so that it can finish its processing involving a shared resource faster so that other processes will be able to use it.

As explained above, neither Bertoni nor Wilner teach or suggest Applicants' claimed method of identifying a process that is an UPGRADER of a resources and granting the UPGRADER write access to the resource, no does Wilner teach or suggest boosting the priority of processes that have upgraded from a read to a write hold on a lock that manages a shared resource. Therefore,

despite the fact that claims 10, 11, 20, 21, 31, and 32 depend, directly or indirectly, on allowable independent claims, these claims are also allowable because the art of record (Bertoni in view of Wilner) does not teach or suggest their claimed limitations. Therefore, claims 10, 11, 20, 21, 31, and 32 are allowable over Bertoni in view of Wilner.

Conclusion

As a result of the foregoing, it is asserted by Applicants that the remaining claims in the Application are in condition for allowance, and Applicants respectfully request an early allowance of such claims.

Applicants respectfully request that the Examiner contact the Applicants' attorney listed below if the Examiner believes that such a discussion would be helpful in resolving any remaining questions or issues related to this Application.

Respectfully submitted,

By Joseph T. Van Leeuwen

Joseph T. Van Leeuwen
Attorney for Applicant
Registration No. 44,383
Telephone: (512) 301-6738
Facsimile: (512) 301-6742